**E26**

|  |  |  |
| --- | --- | --- |
| **Design** | **PROS** | **CONS** |
| Design1: Store one type of coordinates using a single pair  of instance variables, with a  flag indicating ,,.,hid1 type is  stored | * High Efficiency(memory) * Can store both Polar Coordinate and Cartesian Coordinate * Do not need to convert | * Require computation to convert Polar Coordinate and Cartesian Coordinate if needed |
| Design 2: Store polar  coordinates only | * directly * don’t need to convert when storing only polar coordinates | * Low efficiency * Large positional error |
| Design 3: Store Cartesian  coordinates only | * Good efficiency. * Easy to implement * It is easy to calculate distances or area problems. | * Consumes a lot of memory * Not suitable for solving complex problems |
| Design 4: Store both types of  coordinates, using four  instance variables | * High efficiency * Can solve complex problems in easy ways. * Easy to implement * Can store both type of points | * Need to write lots of complex codes * Consumes the most memory * Slower for initializing a point |
| Design 5:Abstract superclass  with designs 2 and 3 as subclasses | * Easy to code * Easy to detect error * Can store both type of points | * More consumption of memory than design 1,2,3 but less than 4 |

**E28**

From the result of experimentation, the design 1 which directly writes the code in one file is averagely faster than the abstract method. In other words, the efficiency of the design 1 is higher than the design 5. The efficiency of the result is exactly the same as the hypotheses we made in E26. The design 1 is more efficient than design 5.

**E29 E30**

The following table shows the Average computation time of Design 1 and Design 5 took.

The loop took about 35 seconds.

|  |  |  |
| --- | --- | --- |
|  | **Design1** | **Design5** |
| Initializing a point with Cartesian Coordinate | 64ns | 110ns |
| Initializing a point with Polar Coordinate | 64ns. | 64ns |
| Getting Cartesian Coordinate from Cartesian Coordinate | 15ns | 28ns |
| Getting Polar Coordinate from Cartesian Coordinate | 33ns | 15ns |
| Getting Cartesian Coordinate from Polar Coordinate | 27ns | 15ns |
| Getting Polar Coordinate from Polar Coordinate | 15ns | 21ns |

In order to run a performance analysis, we created a new class called “runTime” to generate large number of random points, which are stored in PointCP1 and pointCP5. (The file runtime.java come with the assignment file.)

Comparing & Discussion:

Form the table above, we can find that the time that initialize a point with Polar Coordinate and Cartesian Coordinate are same in the Design1. That may be because it can store with the Polar or Cartesian Coordinate directly (without conversion). Also, the time of getting coordinates from the original type (i.e. getting Polar Coordinate form Polar Coordinate) is less than the time of getting coordinates form a different type (i.e. getting Polar Coordinate form Cartesian Coordinate). That is because it took time to do the conversion.

Focusing on the Design 5, it is different from the Design1. Design5 can only store one type of coordinate. The store type depends on the initialized coordinates, but it can compute the point into the other type.

By comparing the Design1 and Design5, we found that the Design1 can store the Polar Coordinate and the Cartesian Coordinate. However, the Design5 only store one type of coordinate, and it depends on its subclass Design2 and Design3. Also, by comparing the converting/computing part, we found that the when getting coordinate types form the other, Design5 takes less time than Design1. That is because the Design1 is doing the conversion and storing while the Design5 is computing.